Configuration
–
Mode d'emploi

Doozer

OLIVIERI Thomas
SAKHI Mohamed

Sommaire

1.Configurer le serveur....3

2.Compilation....4

3.Site Internet....6
- Accueil....6
- Erreur d'identification....6
- Succès d'identification....7
- Messages....8
- Les musiques....9
- Recherche par artiste....9
- Recherche par album....10
- Recherche par genre....11
- Recherche par playlist....11
- Recherche par dossier....12
- Une de « mes » playlists....13
- Gestion de playlist....14
- Les amis....15
- Les listes d'amis....15
- La recherche d'amis....16
- Les préférences....16

4.Rappels et informations utiles....17

5.Remarques supplémentaires....18

1. Configurer le serveur

La configuration est ***indispensable*** pour le bon fonctionnement de l'application sur une machine différente.

Pour ce faire : projet ***EJB*** > package ***server*** > classe ***ServerConfig***

```
13  public final class ServerConfig {
14
15      // LOCAL PATHS
16      public static final String TESTFILES_PATH      = "C:\\Dev\\NetBeansProjects\\Doozer\\Doozer-ejb\\src\\java\\server";
17      public static final String ADMINS_DATA_PATH    = TESTFILES_PATH + "\\admins.csv";
18      public static final String USERS_DATA_PATH     = TESTFILES_PATH + "\\users.csv";
19      public static final String PLAYLISTS_DATA_PATH = TESTFILES_PATH + "\\playlists.csv";
20      public static final String SERVER_LOCAL_PATH   = "C:\\Dev\\glassfish3\\glassfish\\domains\\domain1\\docroot";
21      public static final String MUSIC_PATH          = SERVER_LOCAL_PATH + "\\Music"; //"D:\\Musique"; //SERVER_LOCAL_PATH + "\\Music";
22
23      // Root URL of the music DB
24      public static final int SERVER_PORT = 9090;
25      public static final String SERVER_URL = "http://localhost:" + SERVER_PORT;
26
27      // ADMINS
28      public static final String[][] ADMINS = {{ServerConfig.SUPER_ADMIN_MAIL, ServerConfig.SUPER_ADMIN_PWD},
29                                               {"thom.olivieri@gmail.com", "tomtom"},
30                                               {"mohamedsakhi@hotmail.com", "simoh"}};
31
32      public static final String SUPER_ADMIN_MAIL = "admin@adm.in";
33      private static final String SUPER_ADMIN_PWD = "admin";
34      public static final String MASTER_PLAYLIST = "MASTER_PLAYLIST";
35
36      // Message read or not
37      public static final String MESSAGE_READ_OK = "read_ok";
38      public static final String MESSAGE_READ_KO = "read_ko";
39
40
41      // Samples config
42      public static final int MIN_PLAYLIST_PER_USER = 1;
43      public static final int MAX_PLAYLIST_PER_USER = 3;
44      public static final int MIN_TRACKS_PER_PLAYLIST = 2;
45      public static final int MAX_TRACKS_PER_PLAYLIST = 30;
46
47      // Supported files
48      public static final String[] SUPPORTED_EXTENSIONS = {"mp3", "MP3", DirectoryFileFilter.INSTANCE.toString()}; //, "wav", "WAV", "wma"
49  }
```

Fig. 1 – server.SeverConfig

Ligne 16 : *TESTFILES_PATH* = chemin absolu des fichiers de test (les fichiers CSV).

Ligne 20 : *SERVER_LOCAL_PATH* = chemin absolu du site (utile pour la lecture des mp3)

Ligne 21 : MUSIC_PATH = chemin absolu du répertoire des musiques (peuvent être ailleurs que sur le site mais pas de lecture possible le cas échéant).

Ligne 24-25 : configuration de l'URL du serveur.

Ligne 28 : *ADMINS* = possibilité de rajouter un élément au tableau ({email,password}). Une autre méthode consiste à éditer le fichier « admins.csv ».

Lignes 32-33 : *SUPER_ADMIN* = l'administrateur qui envoie les messages systèmes.

Lignes 42-45 : configuration des jeux de test : ici, tout utilisateur (exceptés les administrateurs) aura dès le départ entre 1 et 3 playlists ; ces playlists auront chacune entre 2 et 30 musiques (cf. méthode ***rand*** de *listener.InitAppli.java*)

Ligne 48 : les extensions de fichiers à rechercher lors du scan de la base de données.

Base de données :

- Nom = Music
- User = app
- Pwd = app

2. Compilation

A la compilation, il est possible de constater en direct l'état d'avancement :

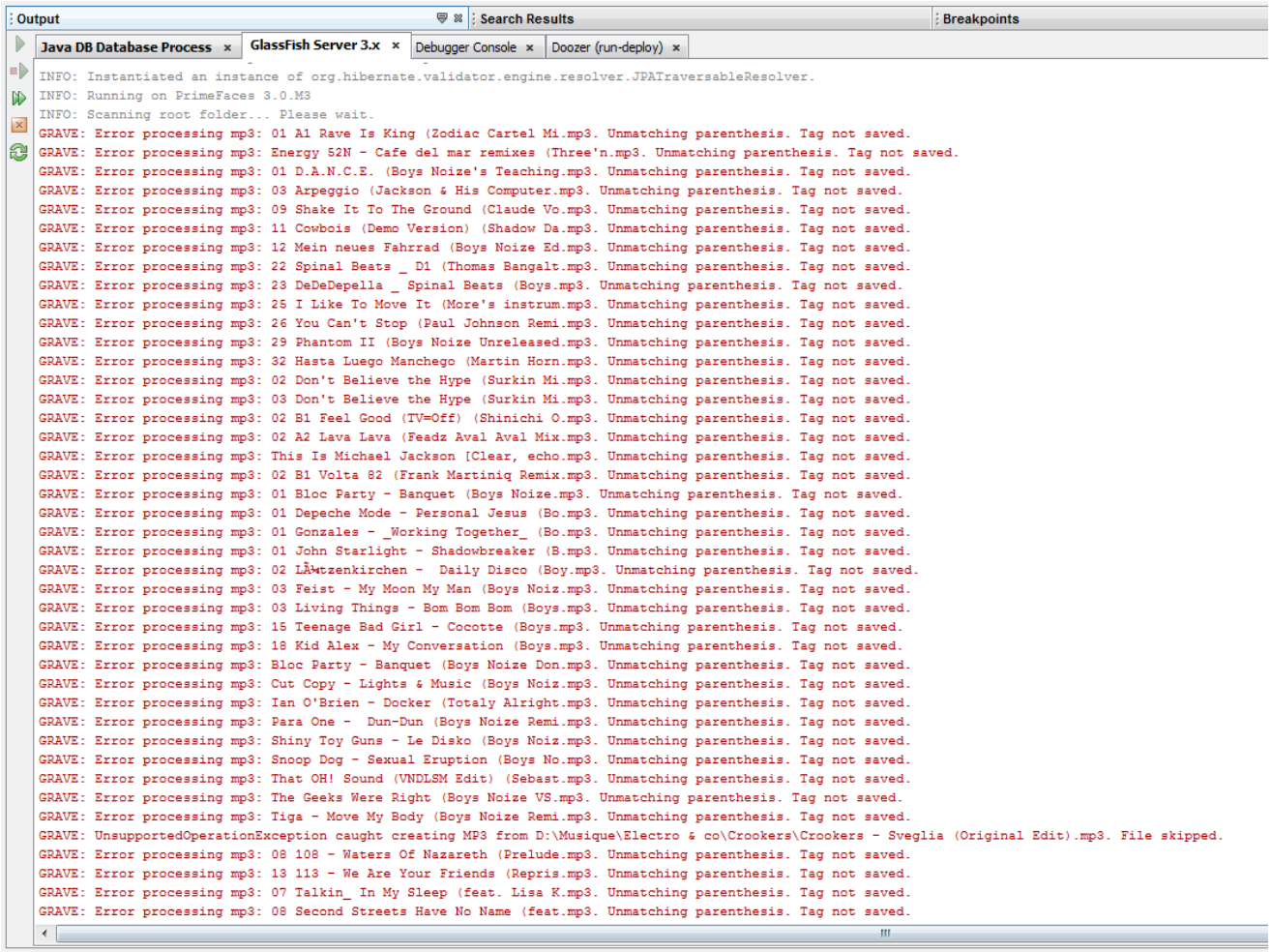

Fig. 2.1 – Début de compilation

Remarques : vous constatez beaucoup de lignes rouges ça n'est jamais bon signe... Mais pour près de 7000 fichiers MP3, « seulement » une cinquantaine de warnings dont 4 importants (*UnsupportedOperationException*).

Une fois la création de la base de données achevée, la compilation s'enchaîne avec la création des utilisateurs et playslists (ces options peuvent être désactivées en éditant la methode « *contextInitialized* » de la classe *listener.InitAppli* du projet WAR) :

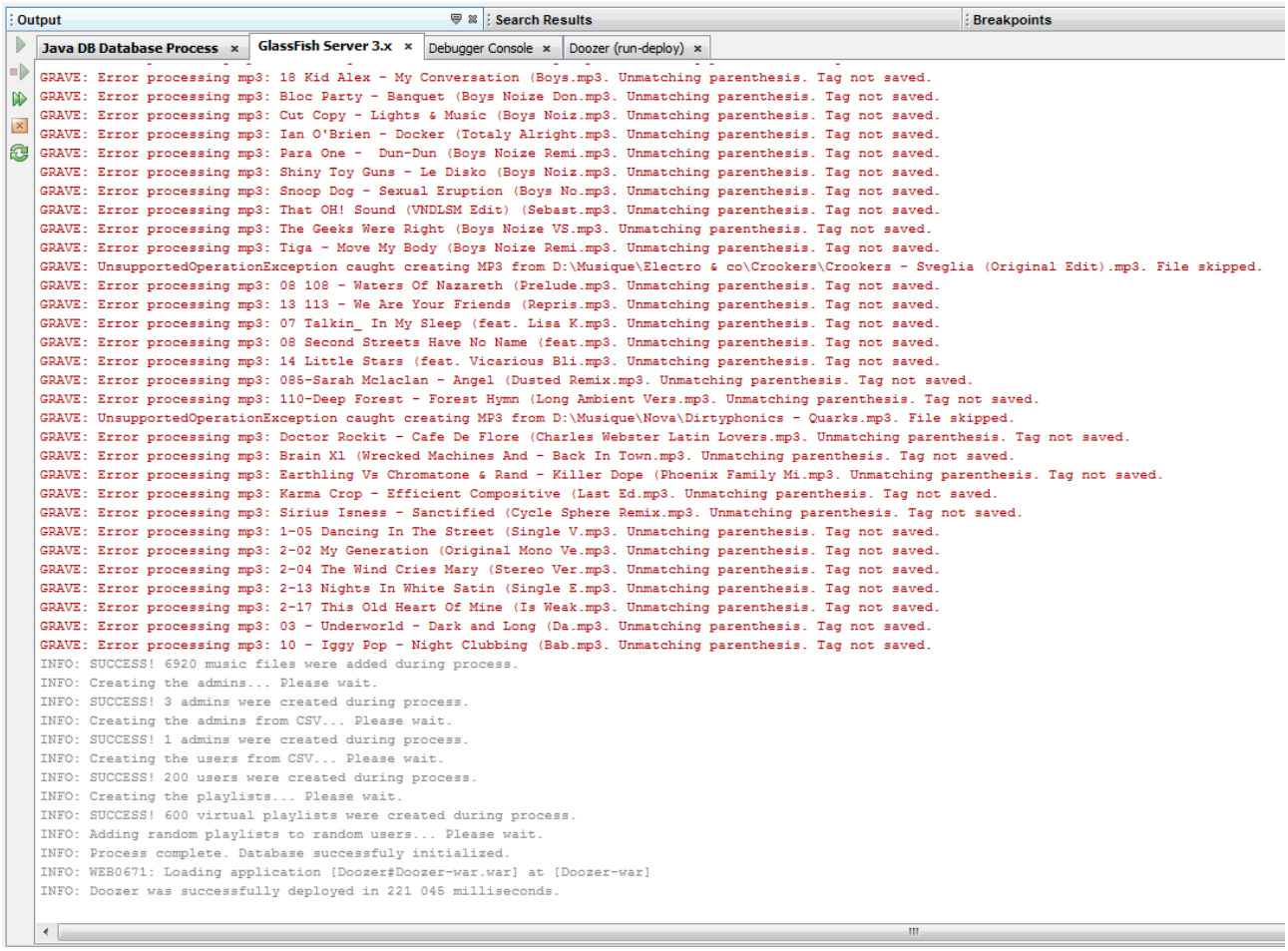

Fig. 2.2 – Fin de compilation

3. Site Internet

Accueil

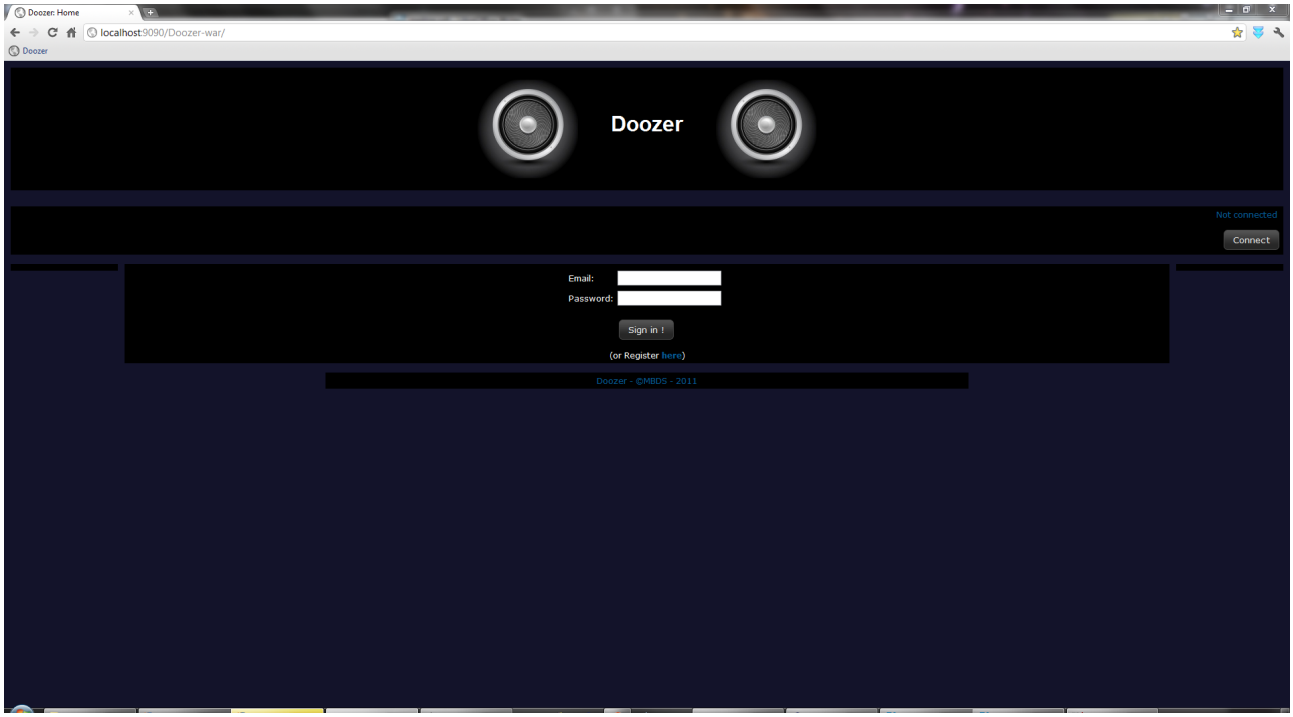

Fig. 3.1 – Accueil

Erreur d'identification

Tout comme pour l'enregistrement (nouvel utilisateur), si le mot de passe est trop court (5 caractères minimum) ou si l'adresse e-mail n'en est pas une, des messages l'indiquent à l'utilisateur.

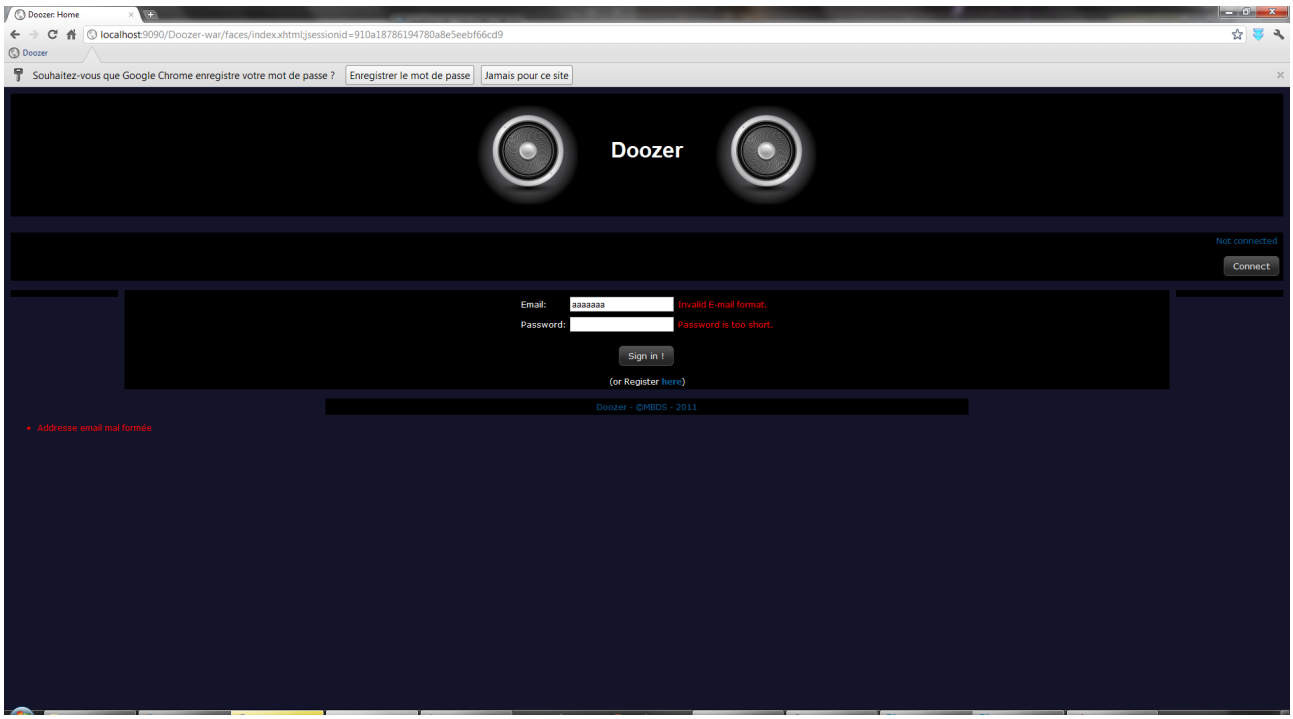

Fig. 3.2 – Erreur d'identification

Succès d'identification

Si l'utilisateur s'est correctement identifié, il est redirigé vers la page correspondante, une playlist aléatoire est chargée, et le *template_connected* entre en jeu :

- à gauche, les options de navigation.
- à droite, les playlists les plus populaires (consulter ses propres playlists n'a pas d'influence).
- en haut, le titre de la page, le lecteur audio et les informations qui vont avec.
- en bas, le bandeau (et le warning expliqué dans la partie « 5. Rappels »)

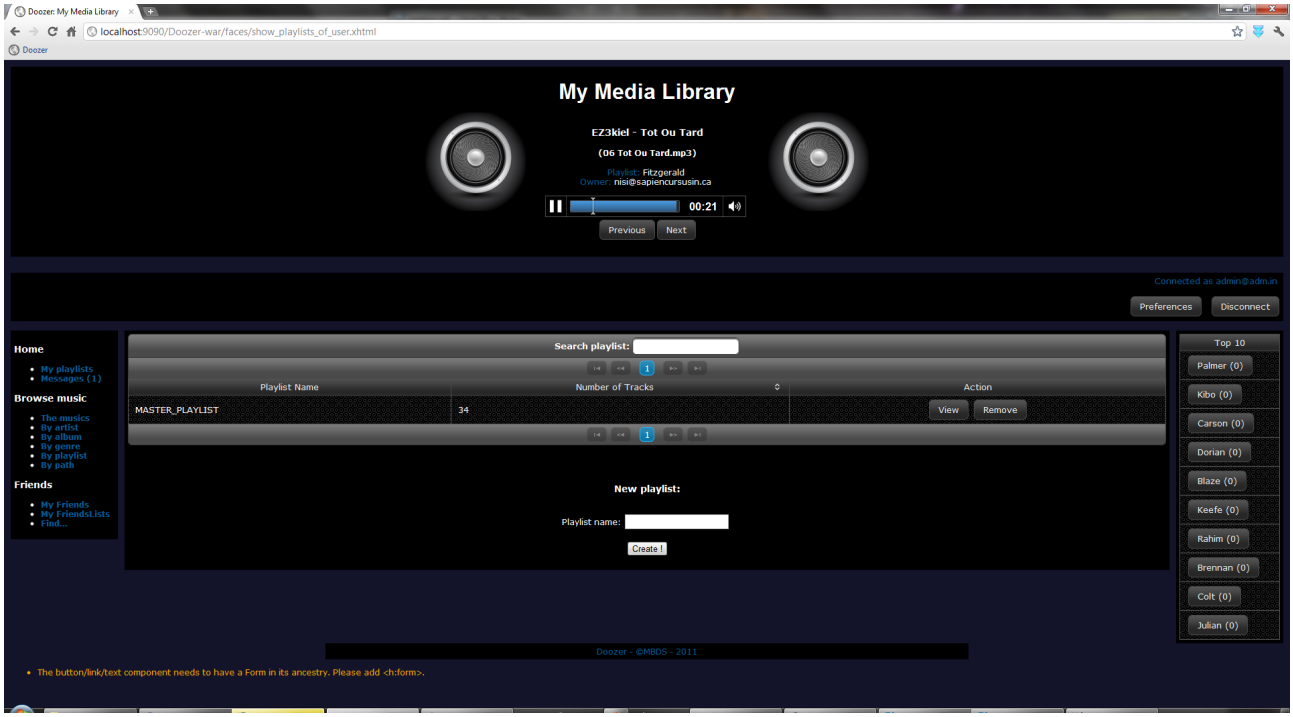

Fig. 3.3 – Succès d'identification

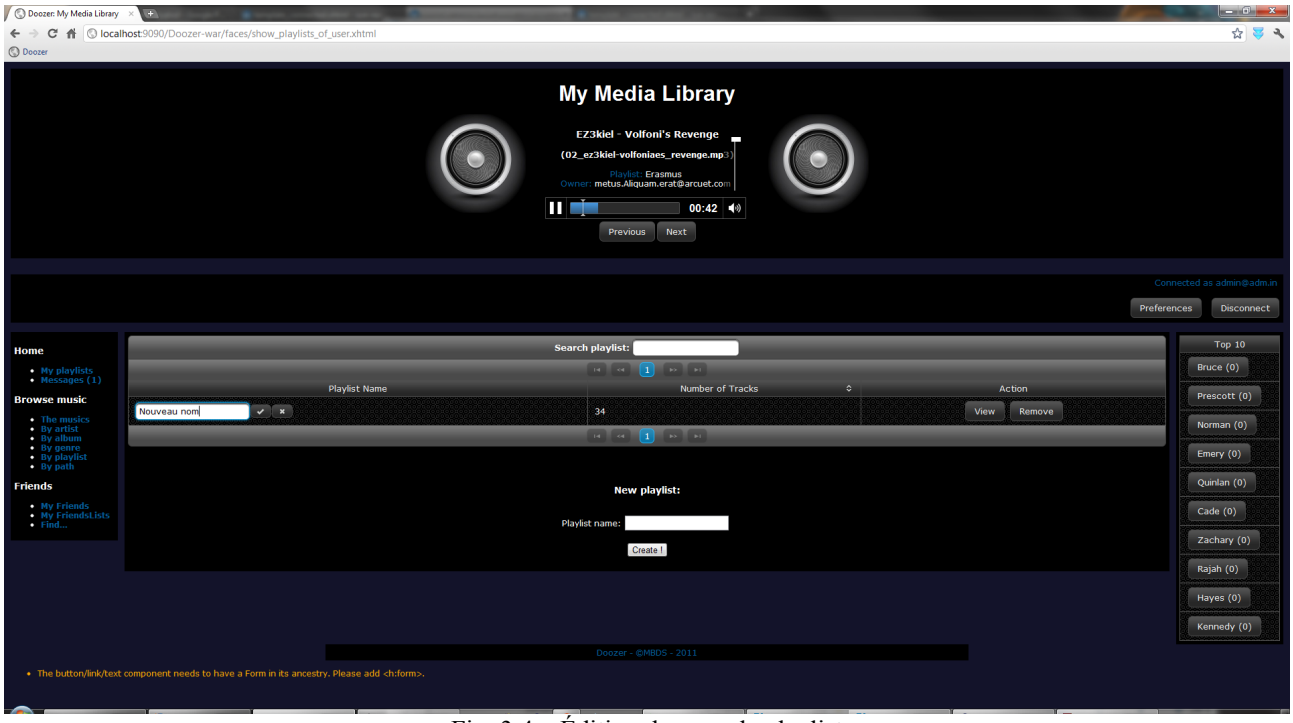

Fig. 3.4 – Édition du nom de playlist

Messages

Chaque nouvel utilisateur reçoit un message de bienvenue.

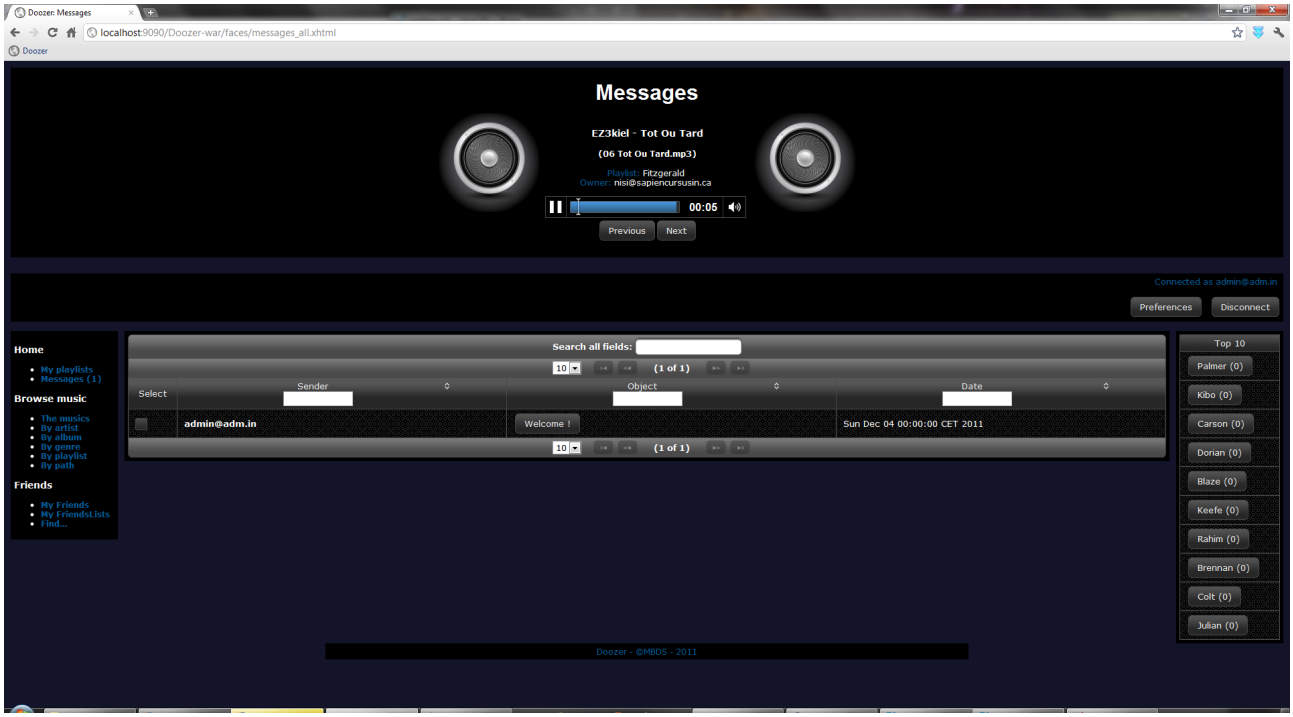

Fig. 3.5 – Messages

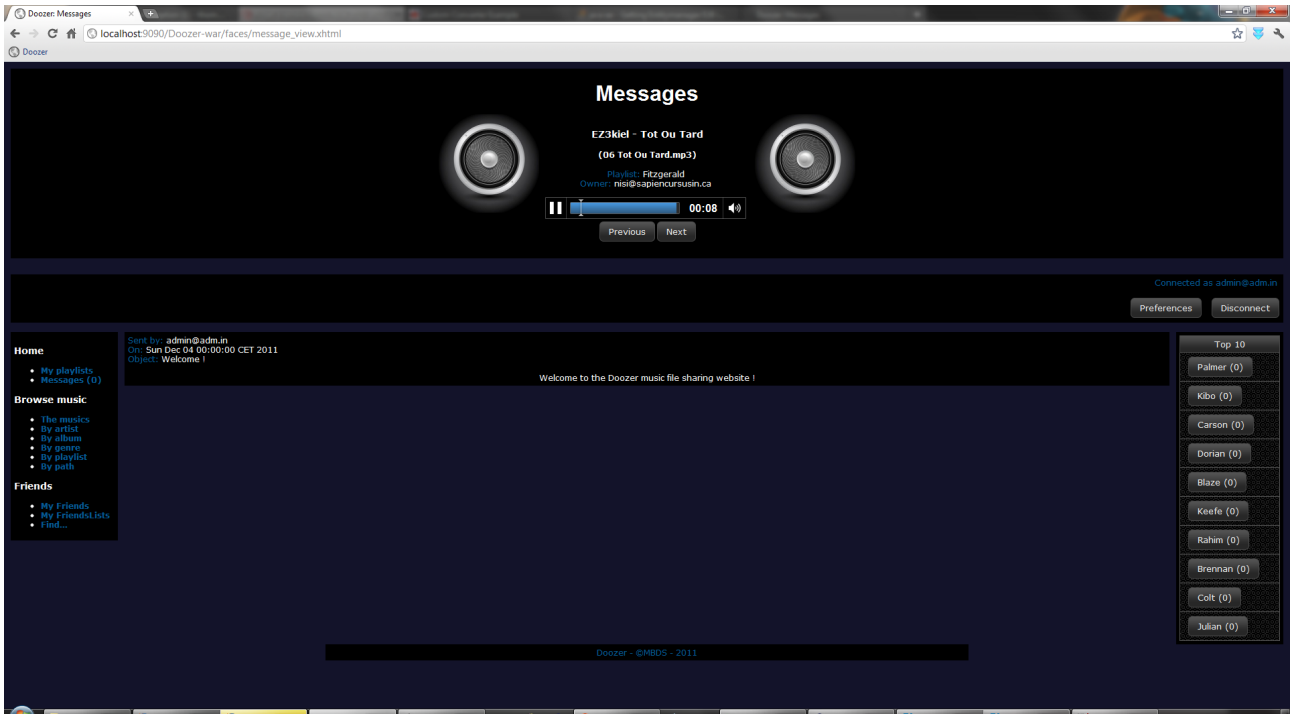

Fig. 3.6 – Un message

Les musiques

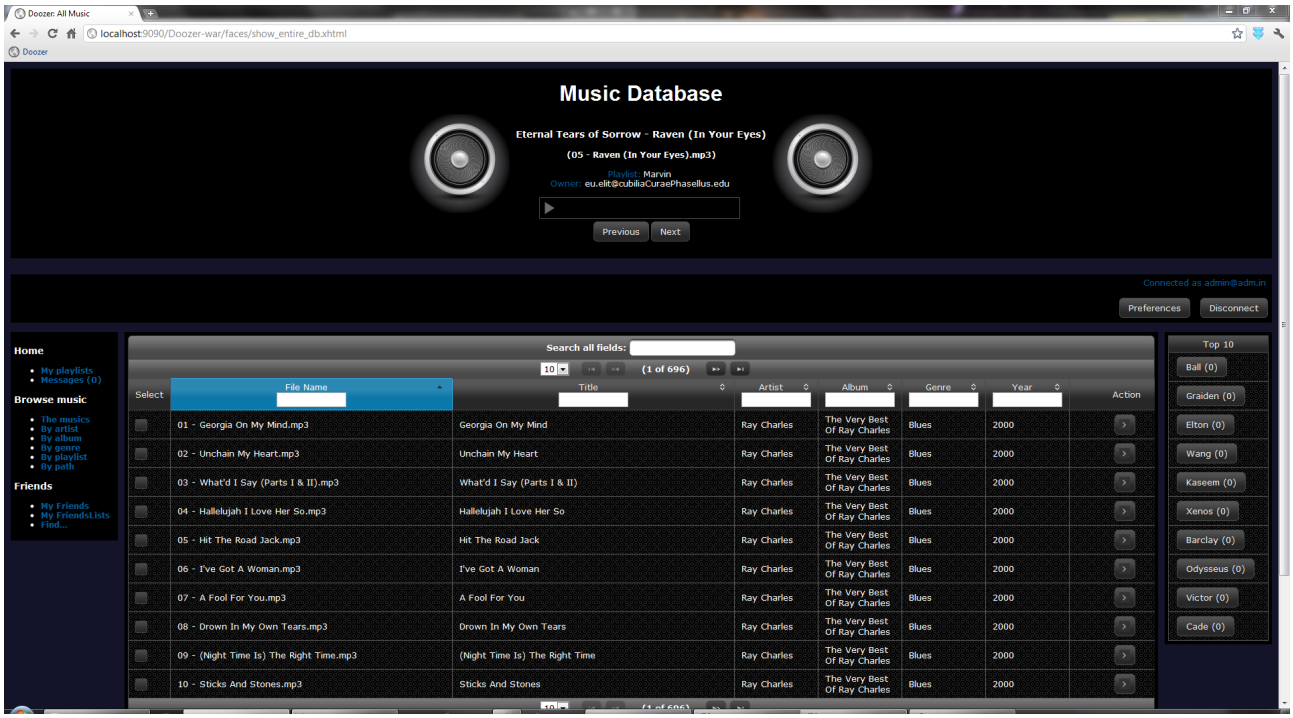

Fig. 3.7 – Toutes les musiques

Recherche par artiste

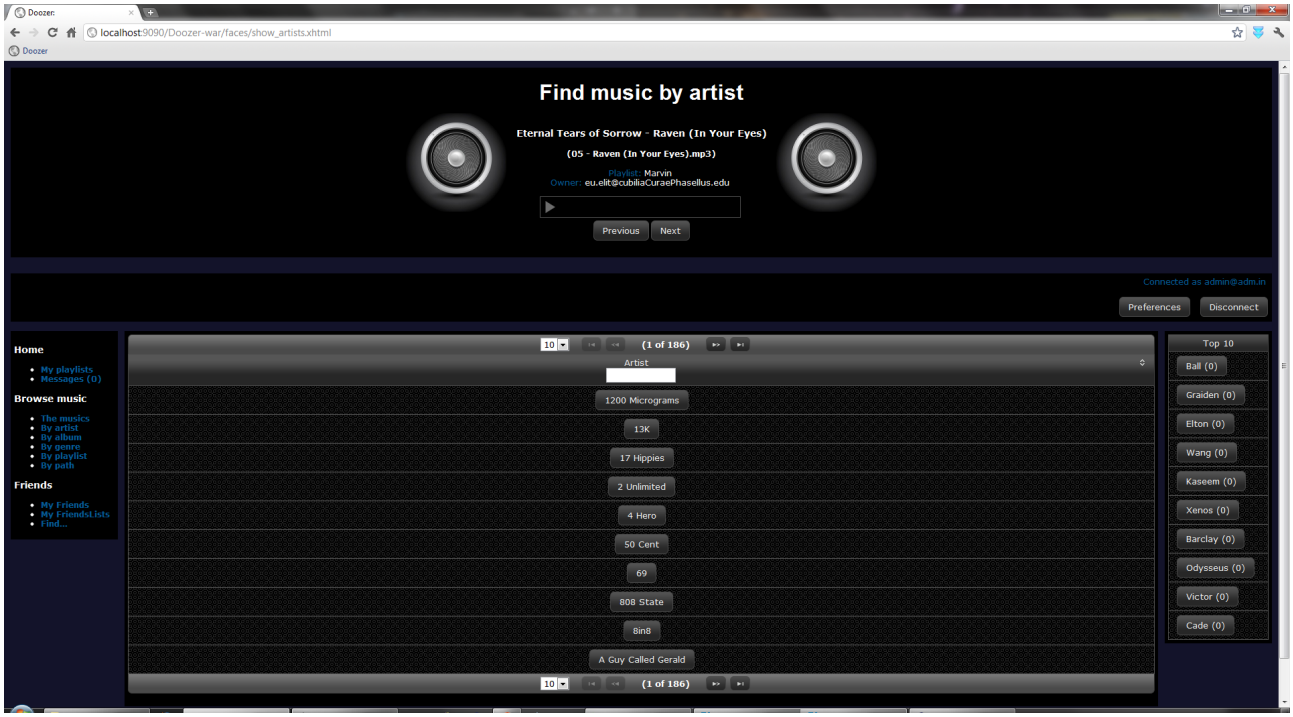

Fig. 3.8 – Recherche par artiste

Recherche par album

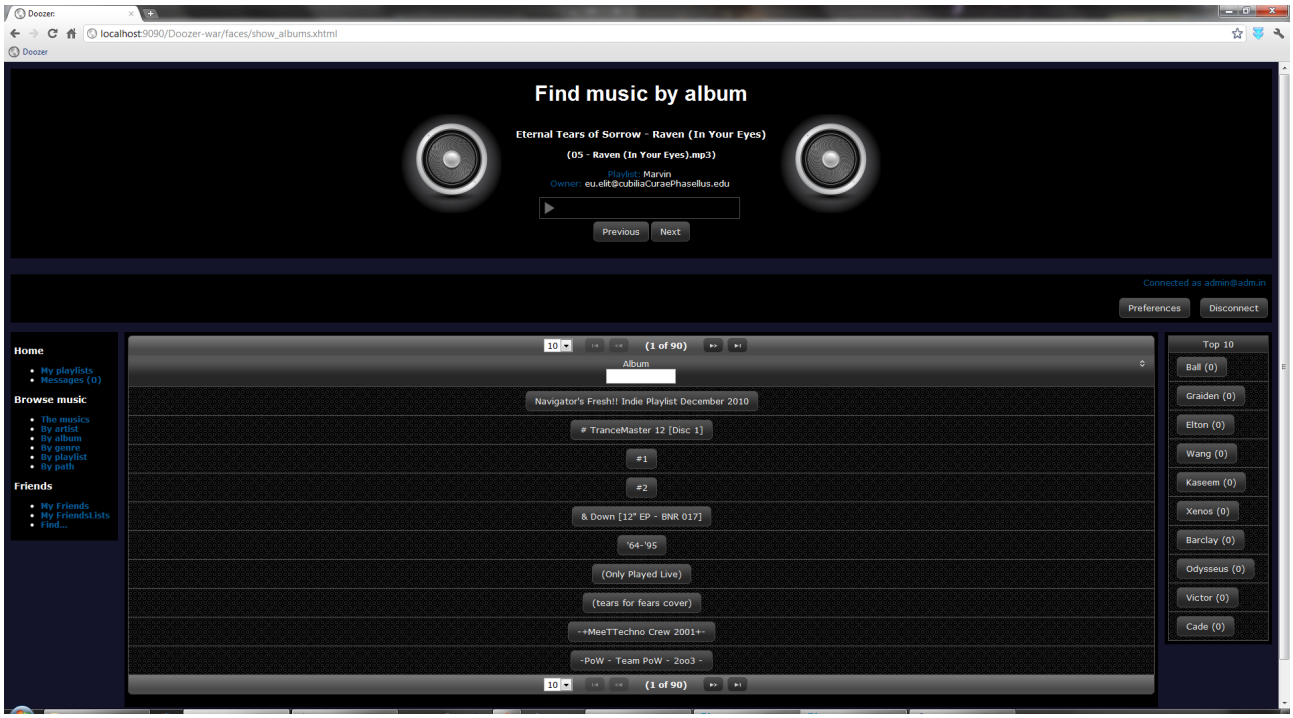

Fig. 3.9 – Recherche par artiste

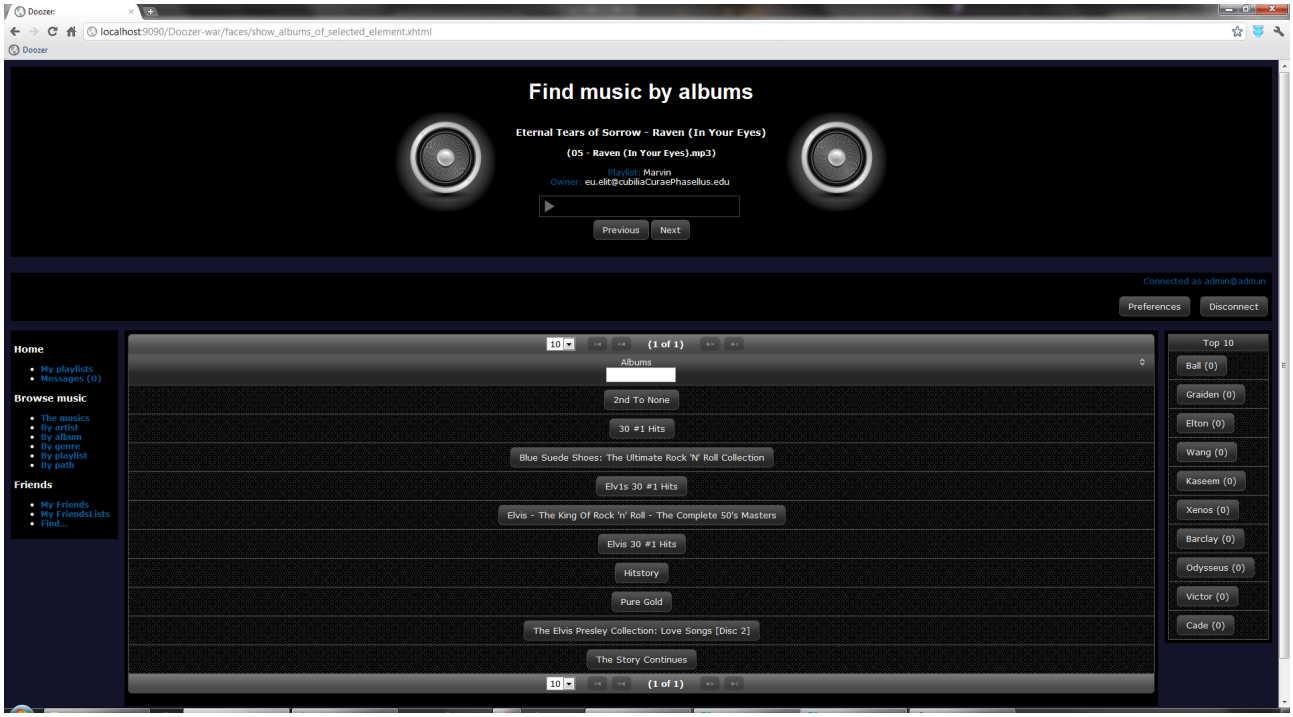

Fig. 3.10 – Affichage des albums de
l'artiste « Elvis Presley »

Recherche par genre

Le genre d'une musique est traité à l'instanciation de l'objet *MusicFile* : un tag ID3 de type « genre » peut être une chaîne de caractères, un entier représentant le genre, ou un entier entre parenthèses. La figure 3.10 représente les genres ordonnés par nom. Les premiers résultats ne sont pas très esthétiques du fait de la malformation du tag. Pour permettre à l'utilisateur de gérer ses propres genres (e.g. « Rock 1 »), nous n'avons pas traité ces cas (i.e. *(entier)genre*).

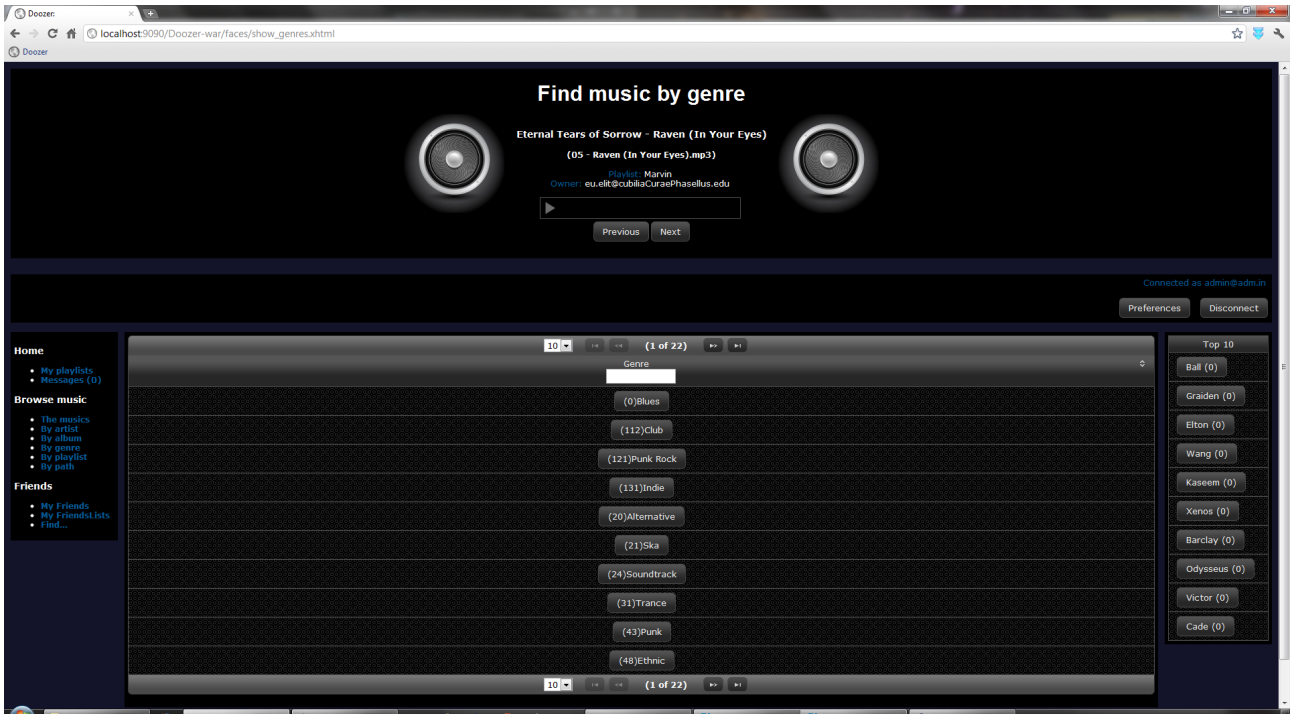

Fig. 3.11 – Recherche par genre

Recherche par playlist

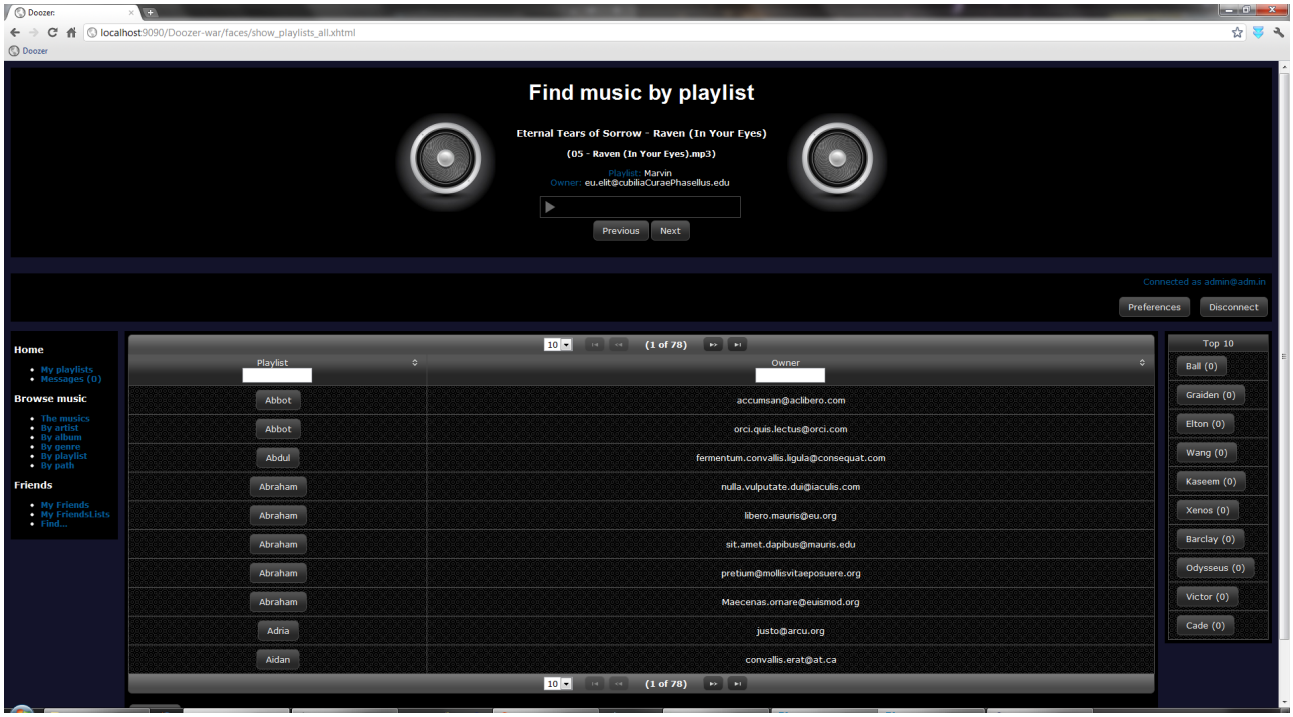

Fig. 3.12 – Recherche par playlist

Recherche par dossier

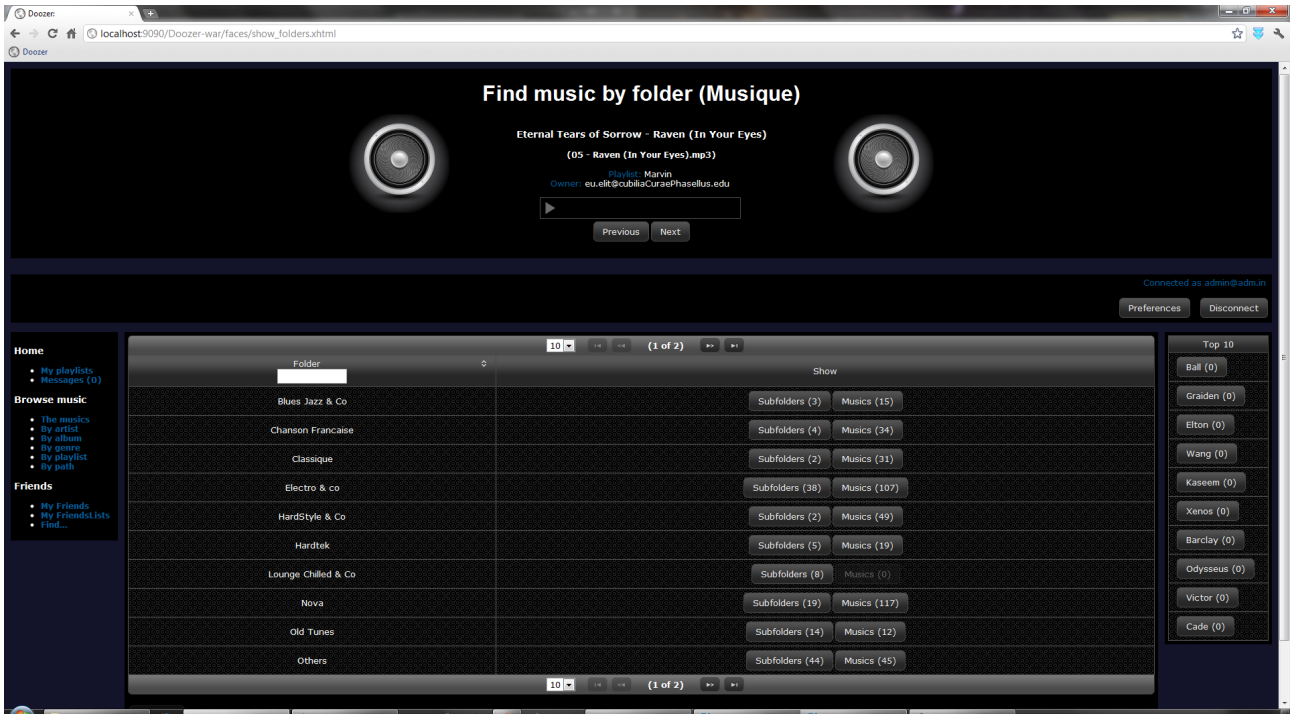

Fig. 3.13 – Recherche par dossier

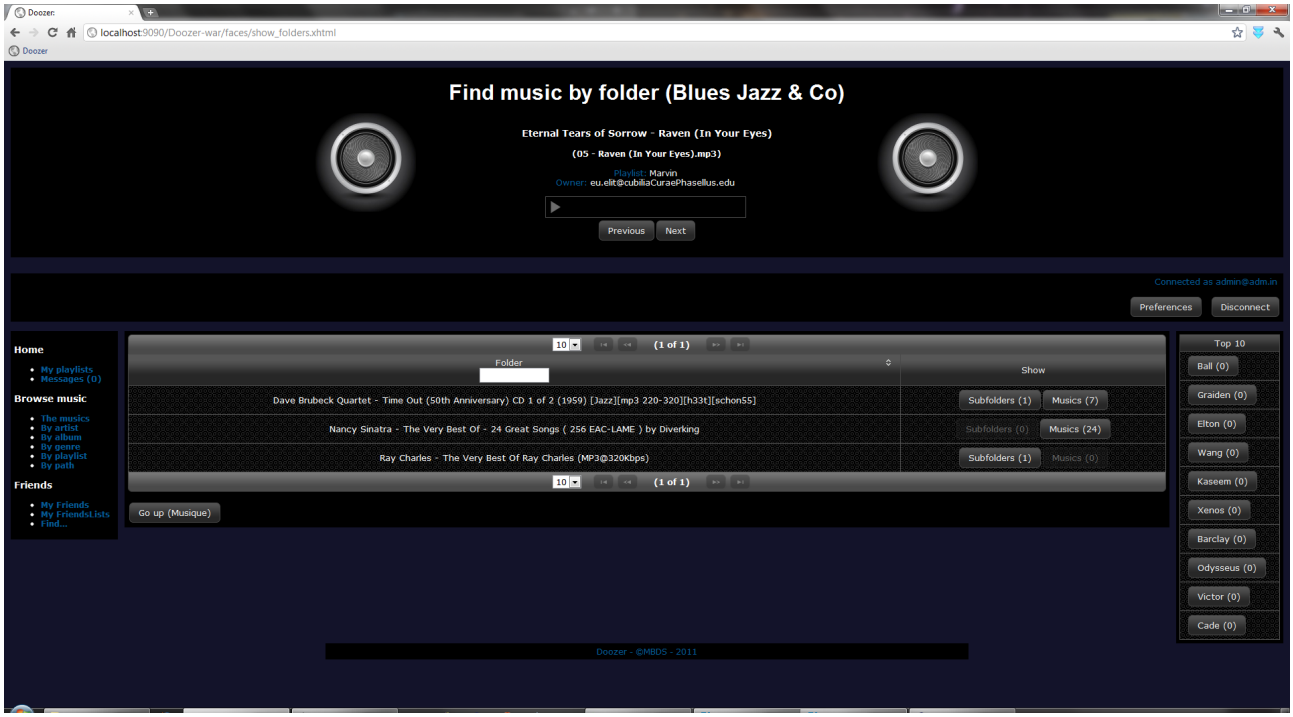

Fig. 3.14 – Affichage des sous-dossiers
du dossier « Blues Jazz & Co »

Une de « mes » playlists

Pour les playlists d'un utilisateurs, de nouvelles options apparaissent. En plus de la lecture, il est possible de supprimer une musique et de réordonner complètement la playlist (penser à sauvegarder une fois les modifications terminées).

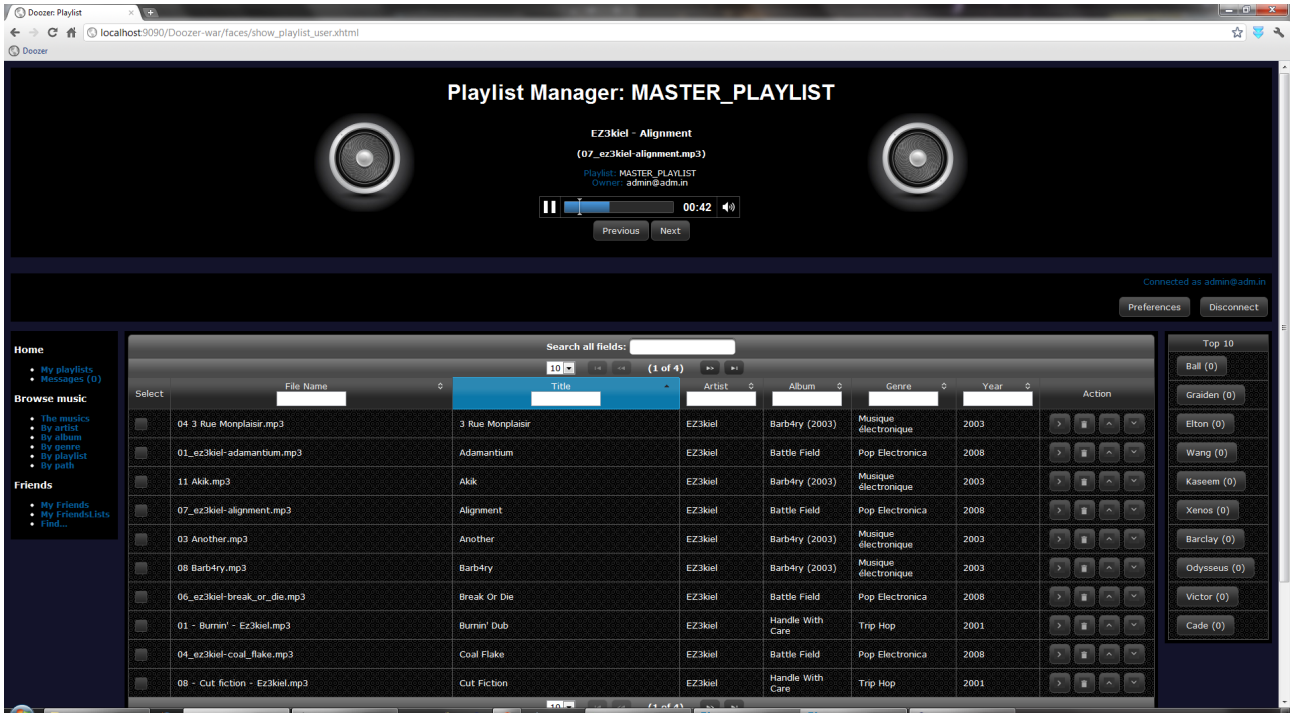

Fig. 3.15 – Une playlist de l'utilisateur

Gestion de playlist

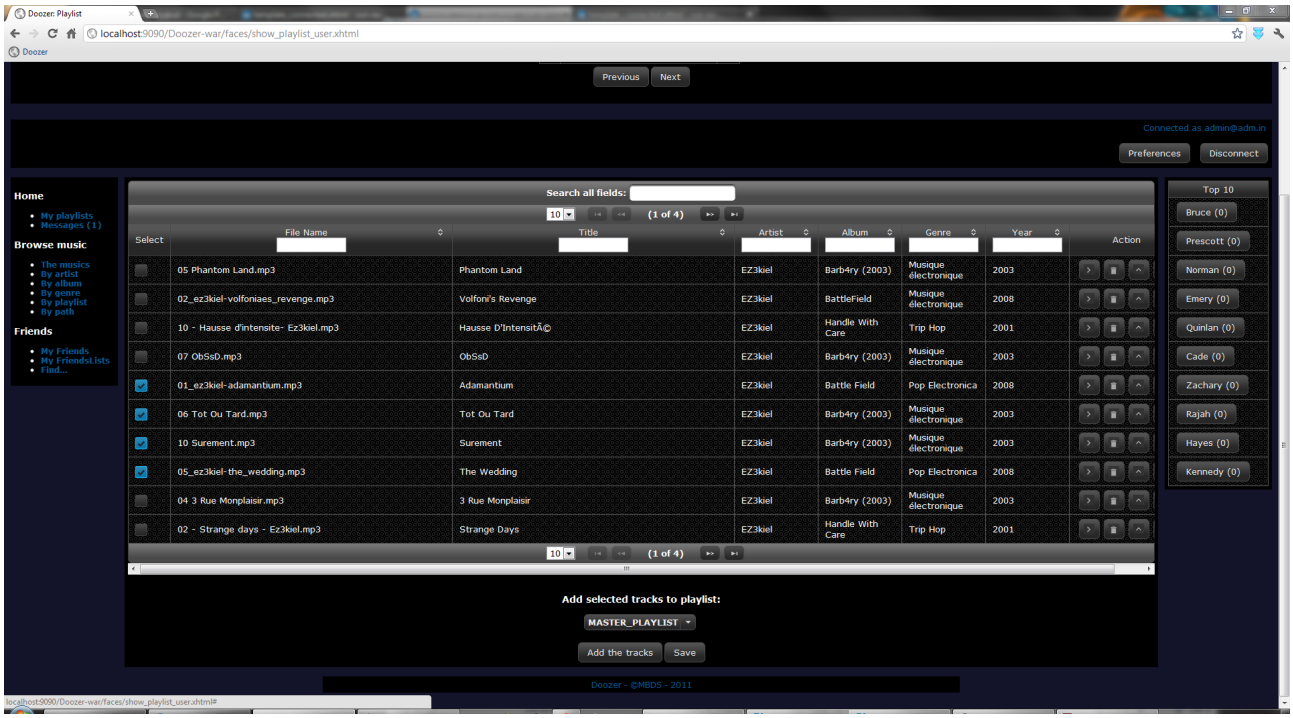

Fig. 3.16 – Ajout de morceaux
à une playlist (d'une playlist à l'autre ou
via la base de données)

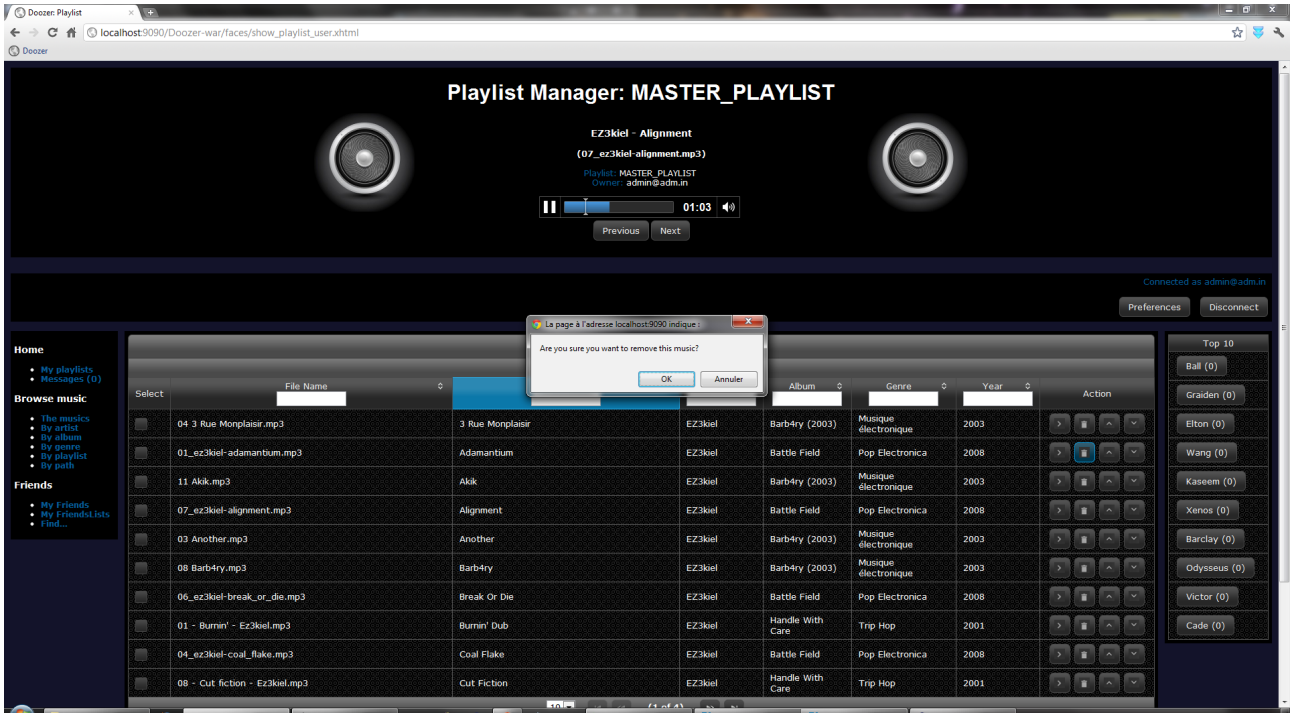

Fig. 3.17 – Suppression d'une musique

Les amis

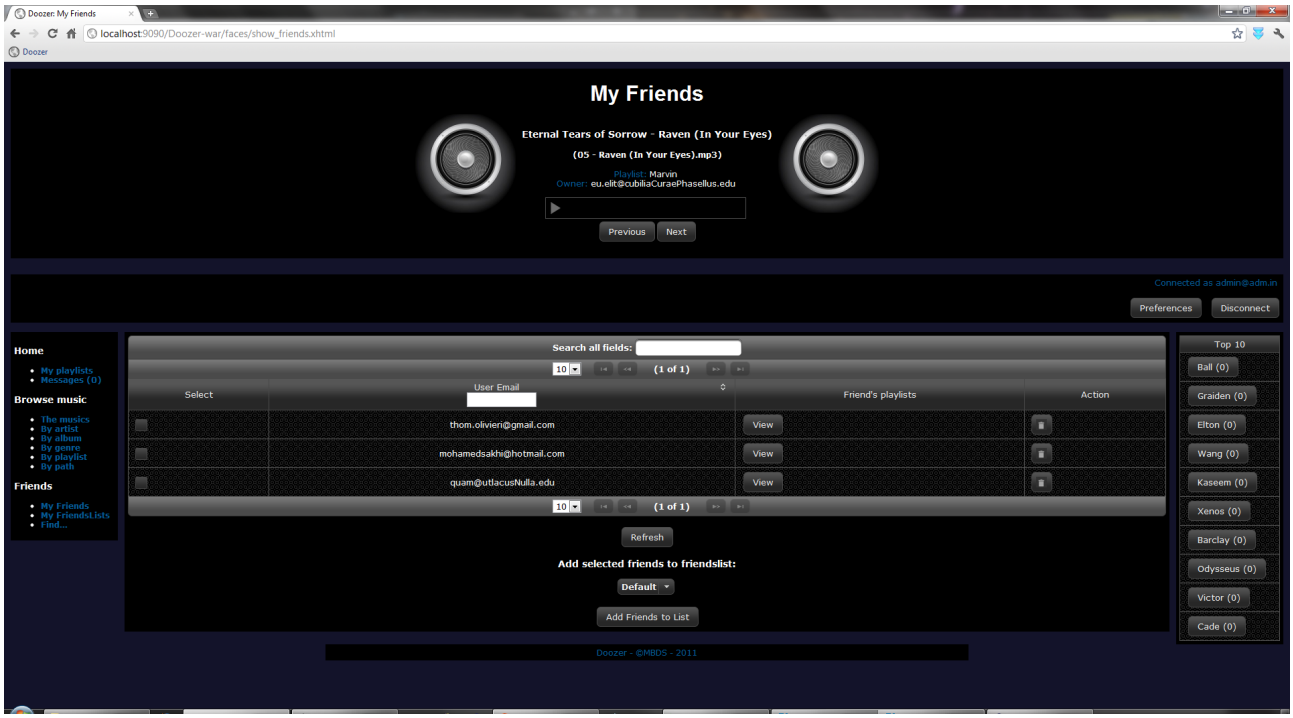

Fig. 3.18 – Amis de l'utilisateur « admin@adm.in »

Les listes d'amis

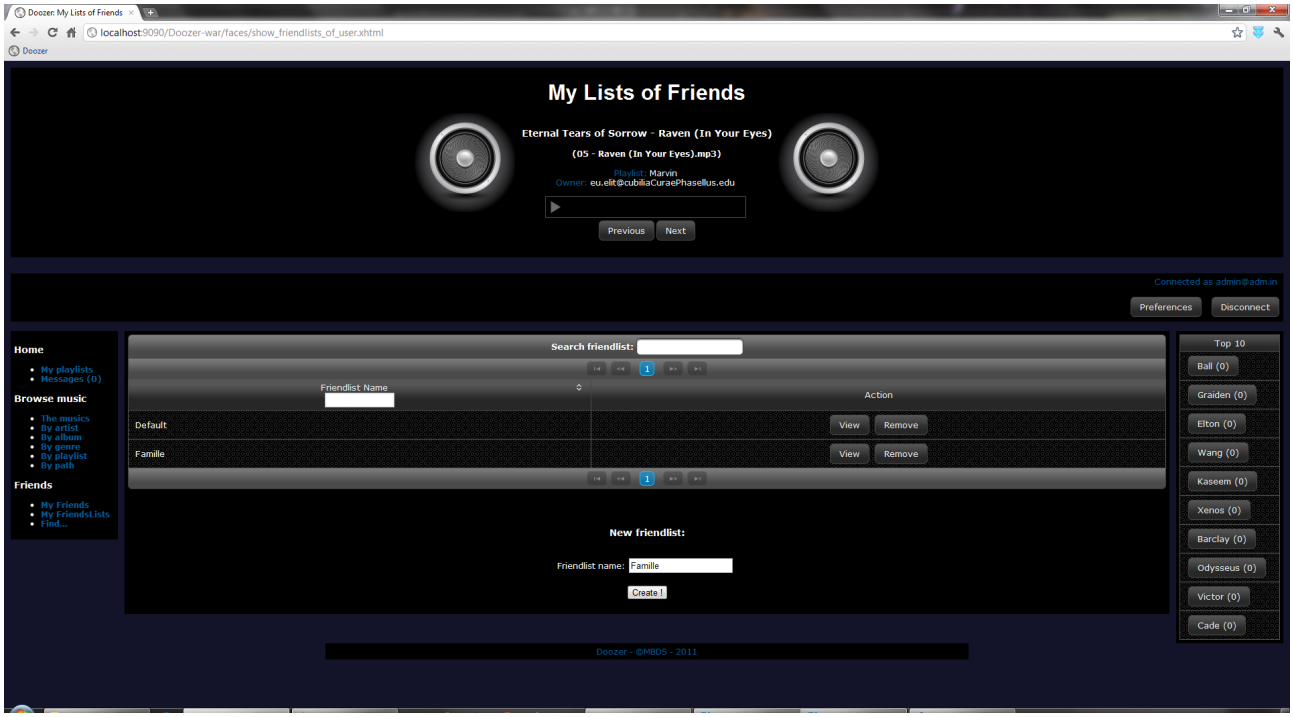

Fig. 3.19 – Les listes d'amis
de l'utilisateur « admin@adm.in »

La recherche d'amis

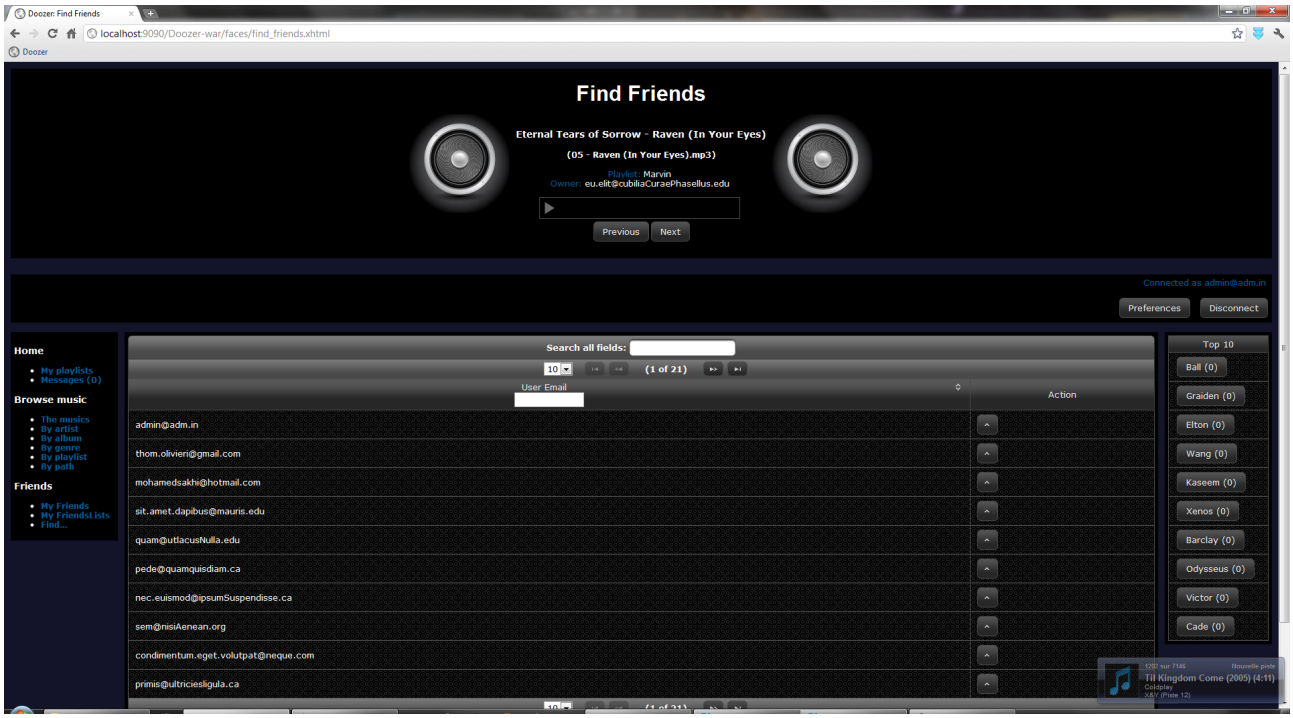

Fig. 3.20 – Recherche d'amis

Les préférences

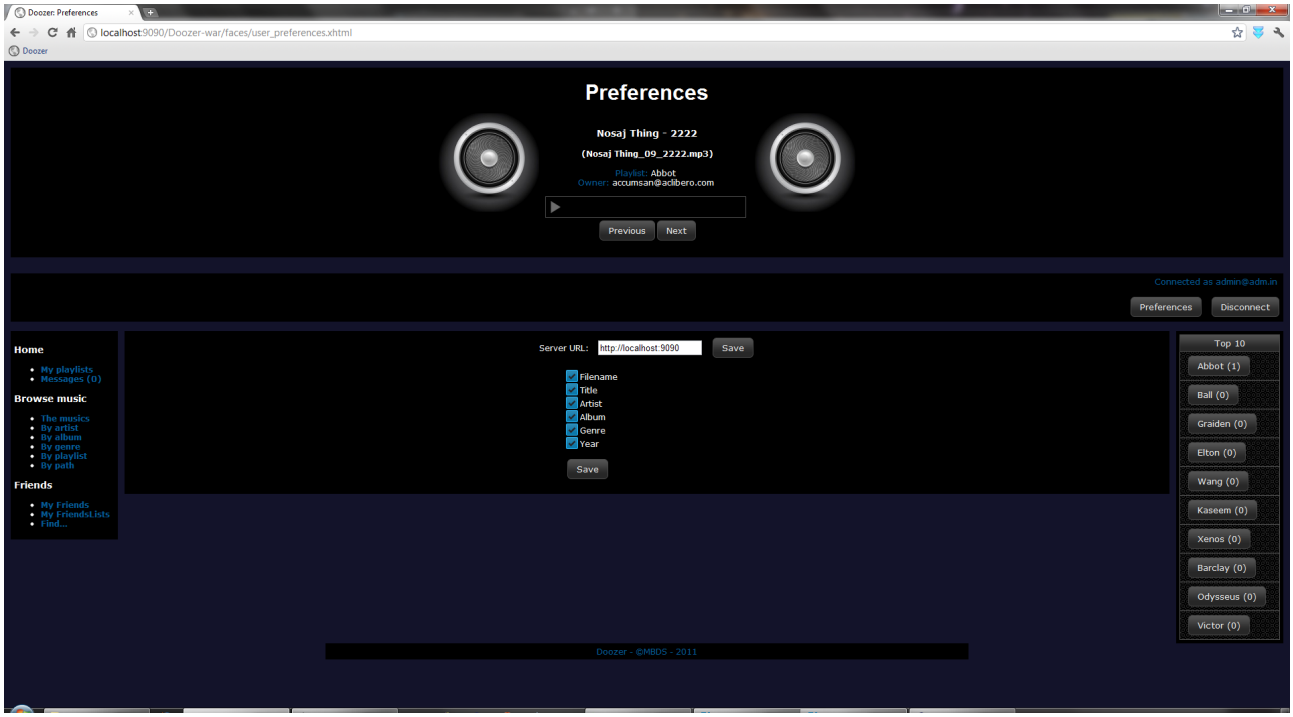

Fig. 3.21 – Préférences

4. Rappels et informations utiles

Base de données :

- Nom = Music
- User = app
- Pwd = app

Points forts :

- intuitif
- bonne base de travail
- code organisé
- jeux de tests

Points faibles :

- inachevé
- gestion des amis
- pas d'ajax
- pas de lecture à la file
- pas d'upload
- code volumineux (en nombre de fichiers) : certains s'y sont perdus, et ne sont toujours pas revenus...)

Pour info :

- 6920 fichiers
- 200 utilisateurs
- 600 playlists (pas nécessairement toutes utilisées)

=> environ 5 minutes de compilation (entre 200 et 450 secondes selon la charge du processeur)

Bug connus :

- La lecture des fichiers musicaux est limitée d'une part par la compression MP3, d'autre part par la prise en charge du navigateur de la balise HTML5 « audio ». Seul Google Chrome © le permet pour cette application.

- Il est possible de naviguer via AJAX en dé-commentant la ligne 15 du *template_connected*. Cela permet de ne pas être dérangé par le rechargement de page et par conséquent le rechargement de lecteur audio (chaque clic recharge l'intégralité de la page ce qui implique une reprise de lecture du début).

- Le warning affiché pour la vue « My Playlists » (warning = « The button/link/text component needs to have a Form in its ancestry. Please add <h:form>. ») a été identifié : il s'agit de l'élément « p:inputText » (de la page « show_playlists_of_user.xhtml ») qui devrait être au sein d'un « h:form » spécifique et personnel. Cependant, en ajoutant ce formulaire, le nom de playlist affiché n'est pas représentatif (nom de l'objet?). Alors que lorsque l'on clique dessus pour l'éditer, le nom correct est affiché... Bug primefaces ? Bref, nous avons décidé qu'il valait mieux un warning avec un nom de playlist correct plutôt que l'alternative.

- Les fichiers MP3 qui possèdent une parenthèse ouvrante dans leur nom de fichier doivent obligatoirement refermer cette parenthèse. Si ce n'est pas le cas, la création de l'objet *MP3* correspondant lève une exception, et le tag ID3 n'est pas sauvegardé.

- Sur certains fichiers (très peu), une exception est levée (à l'instanciation de l'objet *MusicFile* correspondant) et le MP3 n'est pas enregistré.

- La base de données peut ne pas avoir été « bien nettoyée ». Les musiques sont alors doublées...

- Nous sommes conscients que la configuration serveur ne devrait pas se faire au sein d'une classe java. (dans un fichier « .properties »?)

- L'application est en anglais, le code et les commentaires également. Ce choix a arrêté d'en être un le jour où le monde informatique est devenu anglophone... Id est depuis toujours. Aucun d'entre nous n'étant parfaitement bilingue, des erreurs de syntaxe/grammaire ne sont pas à exclure.

5. Remarques supplémentaires

Les screenshots (photos) n'ont pas été pris au cours d'une même session. La cohérence n'est donc pas au rendez-vous (nombre de messages non lus, classement des playlists, etc.). Hélas, ce fastidieux travail devra attendre...